Distributed Network Scheduling

B. J. Clement[1] and S. R. Schaffer[1]

Distributed network scheduling is the scheduling of future communications of a network by nodes in the network. This article details software for doing this onboard a spacecraft in a remote network. While prior work on distributed scheduling has been applied to remote spacecraft networks, the software reported here focuses on modeling communication activities in greater detail and including quality of service constraints. Our main results are based on a Mars network of spacecraft and include identification of a maximum opportunity of improving the traverse exploration rate by a factor of three; a simulation showing reduction in one-way delivery times from a rover to Earth from as much as 5 hours to 1.5 hours; a simulated response to unexpected events averaging under an hour onboard; and ground schedule generation ranging from seconds to 50 minutes for 15 to 100 communication goals.

I. Introduction

This article describes software for distributed network scheduling and presents preliminary results. A remote network of spacecraft can schedule communication in advance along with the rest of their operations. If the spacecraft can make scheduling decisions autonomously, then they can be more responsive to unexpected events. Since the spacecraft will not necessarily be in contact all of the time (due to orbital and resource constraints), scheduling decisions may need to be made in a distributed fashion. The purpose of this software is to automate the distributed scheduling of communications among remote spacecraft autonomously.

This software automatically negotiates the rescheduling of these communications with other spacecraft while respecting constraints with communication resources (such as memory and transceiver availability). It provides an interface for a user or automated process to request communication service and to receive a reservation with updates on the expected or resulting quality of service (QoS).

As shown in Fig. 1, a scheduling node uses Continuous Activity Scheduling Planning Execution and Replanning (CASPER) [1] as the scheduling engine, employs Shared Activity Coordination (SHAC) [2] to coordinate scheduling with other nodes (through middleware), and provides interfaces for requesting communication with the network and for configuring/reconfiguring communication (e.g., determining routing, bandwidth, protocols, etc.). The adaptive communication algorithms in the latter interface could be a middleware function.

[1] Exploration Systems Autonomy Section.

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

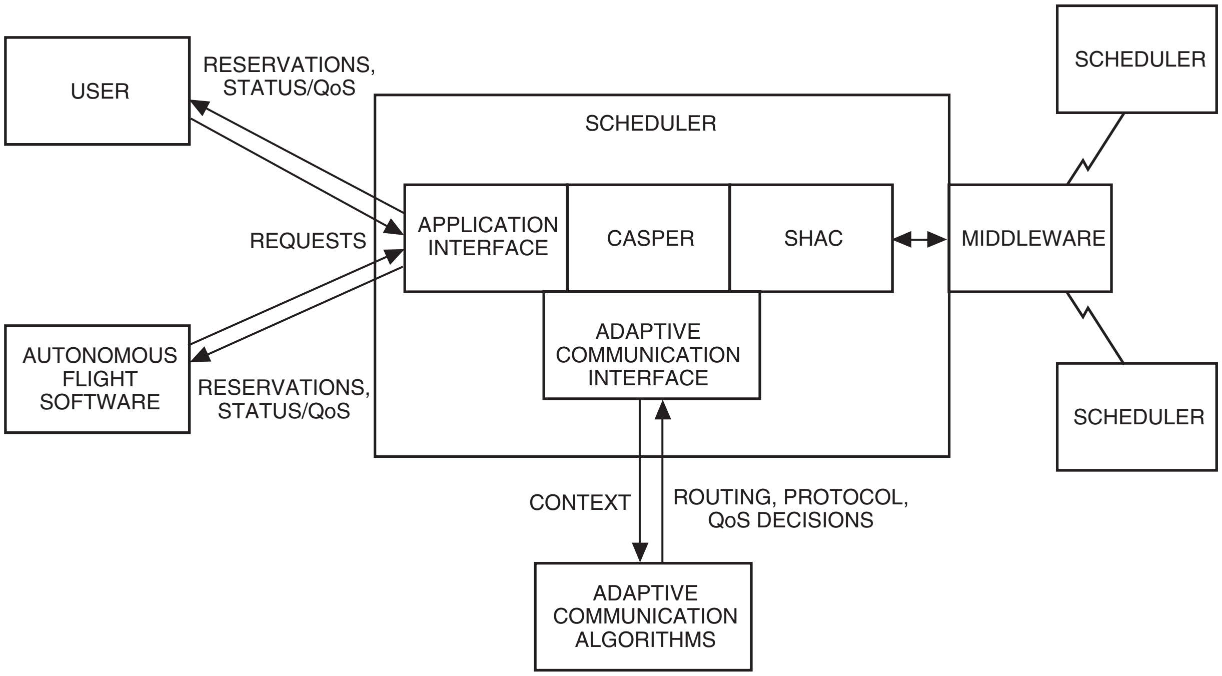

Fig. 1. Distributed network scheduler architecture.

II. Communication Requests

An application or user requests future communication from the network by providing values for the following variables:

- int id: index for tracking
- bool duplex: whether communication will be two way (not supported yet)
- string source: who is sending the data
- string destination: who is receiving the data
- int size: estimate of the size of the data to be sent in kbits
- real bandwidth_min: minimum required bandwidth in kbits/s
- real bandwidth_max: maximum usable bandwidth in kbits/s
- real priority: importance of fulfilling the request (larger numbers indicate greater importance)
- int start_time_min: minimum requested start time of communication
- int start_time_max: maximum requested start time of communication
- int duration_min: minimum needed time duration of initial data transmission
- int duration_max: maximum requested time duration of initial data transmission
- int delivery_time_min: minimum required delivery time
- int delivery_time_max: maximum requested delivery time
- bool progressive: whether data are recreated as they are received (= true) or transmission is only valuable when completed, i.e., all or nothing (= false)

- real loss_overall: maximum percentage loss tolerance of overall data
- real loss_per_block: maximum percentage loss tolerance for any block
- real loss_block_size: size of block for which the loss tolerance is specified
- string protocol: what protocol(s) should be used for transmission and with what options (e.g., CFDP-noack); this string has no generic structure and is to be generated and interpreted by adaptive communications software through an interface

III. Communication Reservation/Status

Upon receiving a request, the network will schedule ("reserve") the communication and reply with the expected quality of service by providing values for the variables listed below. Status is reported using the same construct.

- int id: index for tracking
- real bandwidth_min: minimum expected bandwidth kbits/s
- real bandwidth_max: maximum expected bandwidth in kbits/s
- int start_time_min: minimum expected start time of communication
- int start_time_max: maximum expected start time of communication
- int duration_min: minimum expected time duration of initial data transmission
- int duration_max: maximum expected time duration of initial data transmission
- int delivery_time_min: minimum expected delivery time
- int delivery_time_max: maximum expected delivery time
- real percent_delivered: the percentage of the data delivered or expected to be delivered
- real loss_overall: maximum expected percentage loss of overall data
- real loss_per_block: maximum expected percentage loss for any block
- string protocol: what protocol(s) will be used for transmission and with what options (e.g. CFDP -noack)

IV. Local Scheduling

We use CASPER to schedule communications according to constraints on memory, transceiver availability, and available windows of communication between scheduling nodes (spacecraft). The main activities scheduled are `send`, `receive`, and `relay`, for transmitting, receiving, and relaying data files. Segmentation and reassembly of files are supported for when files are too large to be sent in available communication windows. In addition, scheduling supports cut-through switching, receiving and relaying a file simultaneously when multiple transceivers are available. The timing and duration of activities take into account constraints on communication delay and bandwidth. While quality of service estimates/status are propagated through the network, the scheduler currently does not handle failures, such as over-tolerance data loss.

A. Scheduler Activities

The main activities we use to model data transfer are `send`, `transmit`, `receive`, and `relay`. The `send` activity recursively decomposes into a series of `transmit` activities for segmentation of the file transfer. It also includes a `free_memory` activity following each `transmit`, where the amount of data sent is replenished to a `memory` resource at a time indicated by a `free_type` parameter, which has one of the following values: "`never`," "`on transmission`," "`on delivery`," or "`on_custody_xfer`." If "`on transmission`," the memory is freed at the end of the `transmit` activity. If "`on delivery`," memory is freed at the end of the `receive` activity of the receiving node. The "`on_custody_xfer`" value is intended to support custody transfer protocols that are not yet implemented.

When one node is executing the `send` activity, the receiving node executes a `relay` activity. The `relay` activity decomposes into a `receive` activity. If the receiving node is not the intended destination of the file, then the `relay` activity also decomposes into a `send` activity, routing the data elsewhere.

The `transmit` and `receive` activities are constrained to be scheduled only during available communication windows, which are modeled as states having "`in_view`" and "`out_of_view`" values over time intervals provided by the system designer. These activities also must reserve a transceiver resource from a set provided by the system designer. The adaptive communication algorithms (shown in Fig. 1) provide the assignment algorithm. The `receive` activity also consumes memory in the amount of the data received.

Delay between the start of the `send` and `receive` activities between pairs of nodes is specified through an adaptive communications function. Cut-through switching is implemented in the `relay` activity. This is where a file is received and transmitted simultaneously. The start of the `send` sub-activity (of `relay`) is computed according to the start of the `receive` sub-activity and the data rates of both `receive` and `send` such that data blocks are not sent before they are received. These activities and resources are modeled in the Activity Scheduling and Planning Environment (ASPEN) modeling language [3].

The interface to adaptive communication algorithms (shown in Fig. 1) is simply the provision of many of the dependency functions in the above activities. Again, these functions could be provided in a middleware communication layer.

B. Resources and States

The following resources and states impose constraints on the scheduler:

(1) Memory: Decisions about when to store and delete data transmissions are based on memory availability.

(2) Data: It may be important to keep track of whether particular data files are stored or deleted in case one needs retransmission due to an unexpected failure.

(3) Antenna(s): Spacecraft can communicate with only one (or maybe two) others at a time.

(4) Communication windows: Spacecraft can communicate only when in view of each other.

Obvious resources that are not considered are power and battery energy. We do not consider it to be the network scheduler's role to handle these other resources, and we assume that their safe use is guaranteed by ground operations or an onboard planning and execution system.

C. Metrics

The network scheduler currently reschedules to resolve conflicts, but it can be extended to optimize the schedule according to summed priority of scheduled activities over a time horizon by using ASPEN's optimization framework. The network scheduler itself will be evaluated in simulation according to the time it takes to resolve new or changed requests. This will be compared with current techniques later in Section VI.

D. Adaptive Communication API

The following functions (listed as `dependencies` in the model of the activities) should be implemented in order to decide how to adapt communication for a given context:

- `string antenna <- choose_antenna (my_name, protocol, requested_bandwidth)`: determine which antenna should be used according to the protocol and requested bandwidth
- `string receiver <- route_to (sender, destination, protocol)`: determine to whom data should be routed next
- `real bandwidth <- request_bandwidth (requested_bandwidth, sender, antenna, receiver, bandwidth)`: determine the appropriate bandwidth based on the protocol and the requested bandwidth
- `bool interruptible <- is_interruptible (protocol)`: determine whether or not the transmission can be interrupted and continued later
- `bool progressive <- is_progressive (protocol)`: determine whether or not this protocol allows the file to be created as it is received (otherwise it is sent all-or-nothing)
- `string free_type <- get_free_type (protocol)`: determine when the data can be cleared from memory (i.e., when custody is transferred); valid values are "never," "on_transmission," "on_delivery," and "on_custody_xfer"
- `real percent_delivered <- reply_percent_delivered (protocol, duration, bandwidth, size, percent_delivered)`: determine the percentage of the data expected to be delivered (it may not be possible to deliver all of it)
- `real loss_overall <- reply_loss_overall (protocol, loss_tolerance_overall)`: determine the percentage loss of the overall image according to the protocol and the requested tolerance
- `real loss_per_block <- reply_loss_per_block (protocol, loss_tolerance_per_block, loss_block_size)`: determine the maximum percentage loss of the image per block according to the protocol and the requested tolerance
- `real send_bandwidth <- calc_send_bandwidth (my_name, send_start_time, destination, receiver, antenna, requested_delivery_time, delivery_time_max, send_protocol)`: determine the expected bandwidth to be used by the sender's protocol base on the request
- `string send_protocol <- get_send_protocol (my_name, send_start_time, destination, receiver, antenna, requested_delivery_time, delivery_time_max, send_bandwidth)`: determine the protocol the sender should use for this request

V. Distributed Scheduling

Scheduling is distributed by propagating information through the network to nodes that are affected and by giving each node some level of decision-making authority with respect to local scheduling. We use Shared Activity Coordination (SHAC) [2] to implement this.

A. SHAC Background

SHAC is an interface between planning/scheduling systems, a general algorithm for coordinating distributed planning, and a framework for designing and implementing more specific distributed planning algorithms. Within SHAC, a shared activity is an activity that some set of planners must collectively schedule. It can be a coordinated measurement, a team plan in which planners have different roles, a use of shared resources, or simply an information-sharing mechanism. Planners are coordinated when they reach consensus on the shared activity. Consensus is achieved when they agree on values for members of the shared activity structure, as follows:

(1) Parameters: shared variables (e.g., start time, duration, bandwidth)

(2) Constraints: each planner's constraints on parameter values

(3) Roles: subset of planning agents sharing the activity and what task is assigned to each

(4) Permissions: variables that determine how each planner is allowed to add, remove, and modify a shared activity

Roles determine how an agent participates in the shared activity. For example, a transmit role in a shared communication activity has different resource constraints than the receive role. Roles specify which agents share the activity and can determine permissions and the protocol used to govern the agent's handling of the shared activity. Constraints can specify restrictions the agents have on the values of parameters. By propagating local constraints, agents can make scheduling choices that avoid conflicts with others without knowing the details of their plans. For example, one agent can send a constraint on the time windows of an activity to convey local constraints on scheduling.

Protocols (here meaning distributed planning algorithms) specify how constraints, roles, and permissions of the shared activities change over time and are used to resolve conflicts among the planners. For example, a round-robin protocol rotates permission assignments among the planners, giving them each turns to replan the activity. A delegation protocol assigns and re-assigns agents to roles. Protocols are designed by sub-classing built-in and user-defined protocol classes.

By constructing protocols and modeling the attributes of shared activities, a system designer specifies the autonomy of each agent with respect to decision making and computation. A completely centralized approach gives one agent a role in each activity with full permissions. Decentralization is introduced when agents propagate constraints, when agents fulfill different roles, or when more than one agent has planning/scheduling permissions. The SHAC coordination algorithm (stated simply) is a continual loop of refining/replanning, applying protocols to further modify shared activities, sending shared activity updates to the sharing planners, and integrating updates from others. The planner interface enables different existing planning tools to interact in this framework.

B. Mars Network Model

SHAC must be customized for the particular application domain. Below is an example of a SHAC model file defining how a network of Mars spacecraft interact over communication. This model can be used for other distributed network scheduling applications by simply replacing the names of spacecraft. Further explanation will follow the model.

```
shared_activity mera_communicate {
   start_time_transmit;
   duration_rcv;
   sender;
   source;
   destination;
   bandwidth;
   size;
   requested_bandwidth;
   bandwidth;
   data_priority;
   requested_delivery_time;
   delivery_time_max;
```

```
  delivery_time;
  percent_delivered_overall;
  loss_total_tolerance;
  loss_per_block_tolerance;
  loss_block_size;
  loss_total;
  loss_total_overall;
  loss_per_block;
  loss_per_block_overall;
  prot;

  roles =
    transmit by mera,
    relay by mgs,
    relay by odyssey,
    relay by mex;

  protocols =
    mera NetworkDelegation,
    mgs Subordination,
    odyssey Subordination,
    mex Subordination;

  permissions =
    mera (all),
    mgs (place, detail, lift, abstract, duration, connect, disconnect,
         parameters),
    odyssey (place, detail, lift, abstract, duration, connect, disconnect,
             parameters),
    mex (place, detail, lift, abstract, duration, connect, disconnect,
         parameters);
};
. . . // other similar comm activities between other spacecraft omitted

agent mera {
  planner = AspenPlannerInterface (20, 10, use_system_time, 100.0);
  communication = SocketCommunication ("ports.txt");
  communicator = AspenCommunicator (comm_windows, comm_window_timeline); };
};
. . . // other similar comm agent definitions omitted

protocol NetworkDelegation ();
protocol Subordination ();
```

The shared activity concerns communication between the Mars Exploration Rover-A (MER-A) and an orbiter. The listed parameters are shared between the `transmit` and `relay` activities defined earlier. These parameters are necessary for either negotiation of scheduling or for communicating QoS status. The roles specify which local activity (`transmit` or `relay`) corresponds to each agent (spacecraft) potentially participating. The protocol assignments show that MER-A will negotiate scheduling by delegating the activity to one of the "subordinate" orbiters. The permissions basically limit the orbiters from moving or deleting the activity while allowing MER-A to make any scheduling decision. The agent declaration for MER-A defines which interface is to be used for planning/scheduling and communication. Finally, the protocols are defined with no arguments. The `NetworkDelegation` protocol is sub-classed from `Delegation`, which generically defines an interface for assigning a subordinate role(s) for the shared activity. `NetworkDelegation` assigns the subordinate (relay) role to the spacecraft chosen by the `route_to` dependency function in the ASPEN activity model. The `Subordination` protocol will remove the agent from the shared activity's roles with a specified probability if the agent is unable to successfully schedule the activity locally. This triggers the delegator to assign another subordinate.

VI. Evaluation

The distributed network scheduler enables reactive communications within the context of scheduled operations by autonomously negotiating over communication changes. In addition, the scheduling system can generate schedules using the same negotiation mechanisms. This means that separate missions (such as the many studying Mars) can use this software to collaboratively schedule communications on the ground. A prototype network scheduling system was implemented for communication models of MER-A, MER-B, Mars Global Surveyor (MGS), Odyssey, and Mars Express. We will give experimental results for this application domain after giving more theoretical results illustrating the benefits of adaptive communication for rover exploration that this system enables.

A. Rover Exploration Performance

Here we examine the science return performance of a semi-autonomous rover investigating rocks during a long traverse between sites. We simulate a traverse based on early Mars Science Laboratory (MSL) scenarios where a rover has the ability to autonomously detect rocks of potential scientific interest, downlink images, and investigate based on commands returned after scientists have studied the images. The rover continues along its path and turns back to perform detailed measurements if commanded. Figure 2 illustrates how the rover must traverse the path three times in order to return to the rock. In the worst case, a target rock is identified just after each communication opportunity, causing the rover to traverse the entire distance three times. By providing more communication opportunities through adaptive rescheduling, this backtracking can be reduced, resulting in a theoretical opportunity of threefold exploration speedup.

We simulated traverses with communication opportunities at fixed time intervals and placed rocks along a straight-line path according to a Poisson distribution. The intervals between communication opportunities and total number of rocks were varied for each run. Figure 3 shows a plot of how the rover's exploration speed approaches the optimal as the number of communications opportunities increases with respect to the number of rocks. Adaptive communications allows the rover to take advantage of opportunities that previously were unscheduled. This does not mean that additional bandwidth to Earth is required since only the needed opportunities are taken. By giving more opportunities, the overall

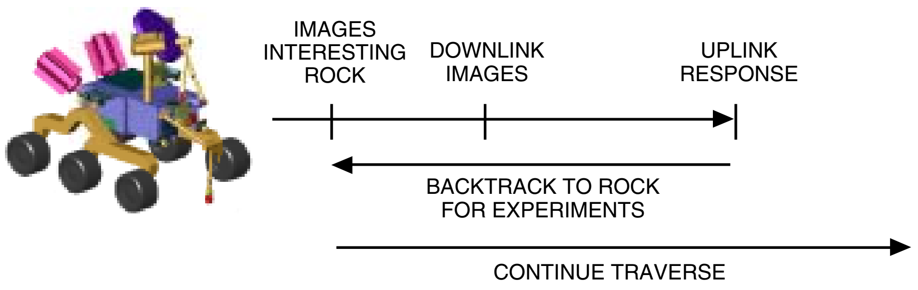

Fig. 2. Rover backtracking to study a rock during a traverse.

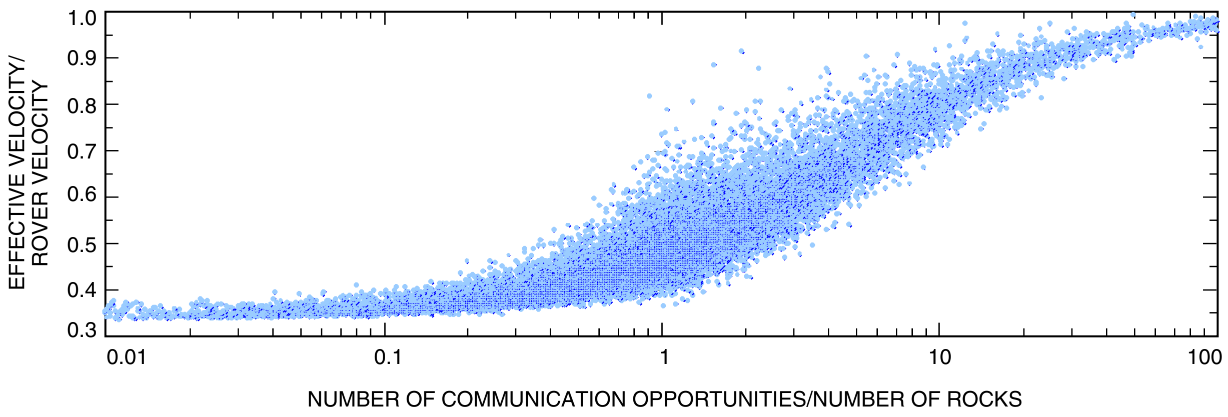

Fig. 3. The effect of increasing communication opportunities on the exploration speed of a rover.

performance is increased. For example, if the communications opportunities are doubled (moving the x-axis ratio from 1.0 to 2.0 in the figure), there is a potential increase from 0.5 to 0.6 (y-axis) in the normalized exploration speed of the rover, resulting in a 20 percent increase in science return.

B. Rover Communication Performance

In some situations, a rover must sit idle while waiting for a response from Earth. For example, using the Rock Abrasion Tool (RAT) can take as long as 9 days because of the uncertainties of executing a long sequence of grinding, drilling, sampling, and measuring and because of the delays of communication in getting new sequences from the ground. Thus, reducing the delay between the need for communication and getting a response is of great interest to Mars missions.

Using information about the duration and frequency of orbiter passes as well as cross-link intervals between orbiters and Earth, we simulated the transfer of data from a rover to Earth with potential routing through orbiters or a direct-to-Earth (DTE) link. Figure 4 shows these communications windows over a little more than a day. MGS and Odyssey passes over the rovers are 10 minutes in duration, and those of Mars Express are 8 minutes. Communication delays are 8 minutes to Earth and 1 second otherwise. Figure 5 shows mean delivery times (in seconds) to Earth (for 1000 simulations per point) for four routing cases based on varying numbers of scheduled communications and possible windows of opportunity. "Static" means that only scheduled links can be used, and "dynamic" means that any window can be adaptively scheduled on the fly. "Direct" means that the rover can use only DTE communications, and "routed" means that it can route through orbiters. Dashed lines show the 95 percent confidence interval, indicating variance. Outliers skew the mean and cause the jaggedness of the plot. Here we assume that all DTE opportunities are scheduled, so the static/direct and dynamic/direct cases are the same.

The difference in the static/routed and dynamic/routed is the performance improvement for rescheduling communications on the fly. For a ratio of one (where all windows are scheduled), performance is the same (as expected), a little over 5000 seconds (about an hour and a half). For fewer scheduled windows, the performance quickly degrades and approaches the DTE plot (at the top) of ranging just below 5 hours. The difference in the static/routed and the DTE plot shows the performance improvement of routing through orbiters as the number of scheduled routing opportunities decreases.

C. Mars Network Experiments

A rover can take advantage of sending data to an over-passing orbiter with minimal negotiation. The orbiter can weigh the priority of the rover's data with the orbiter's current resource needs and decide whether it will agree to relay the data. We have assumed that the orbiters always agree to route the

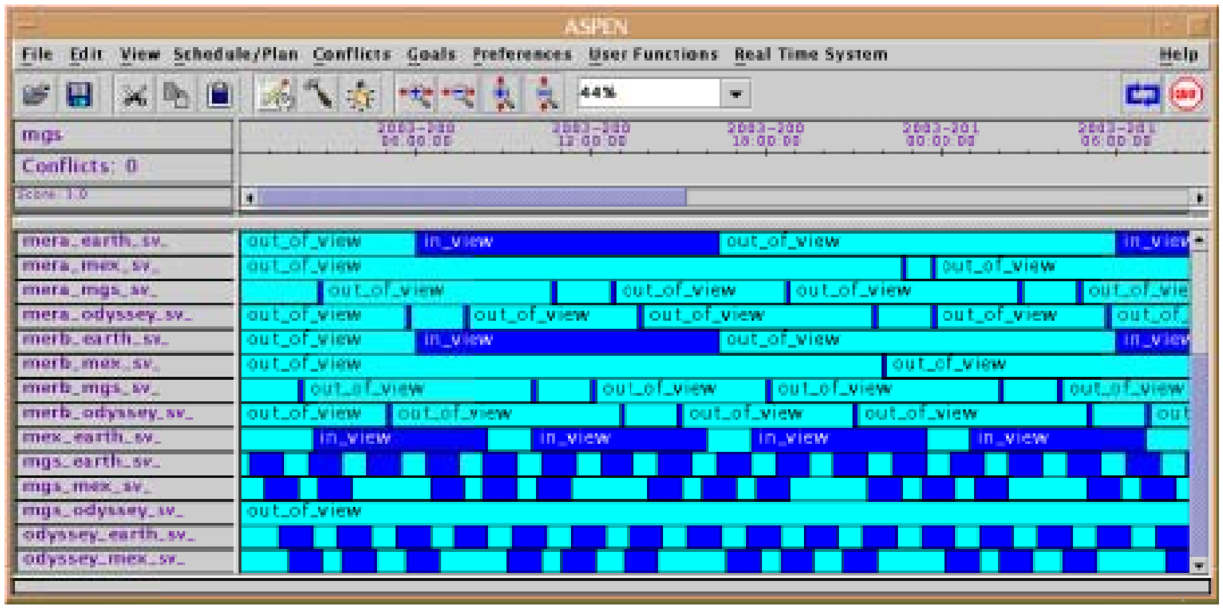

Fig. 4. Communication windows in a Mars network.

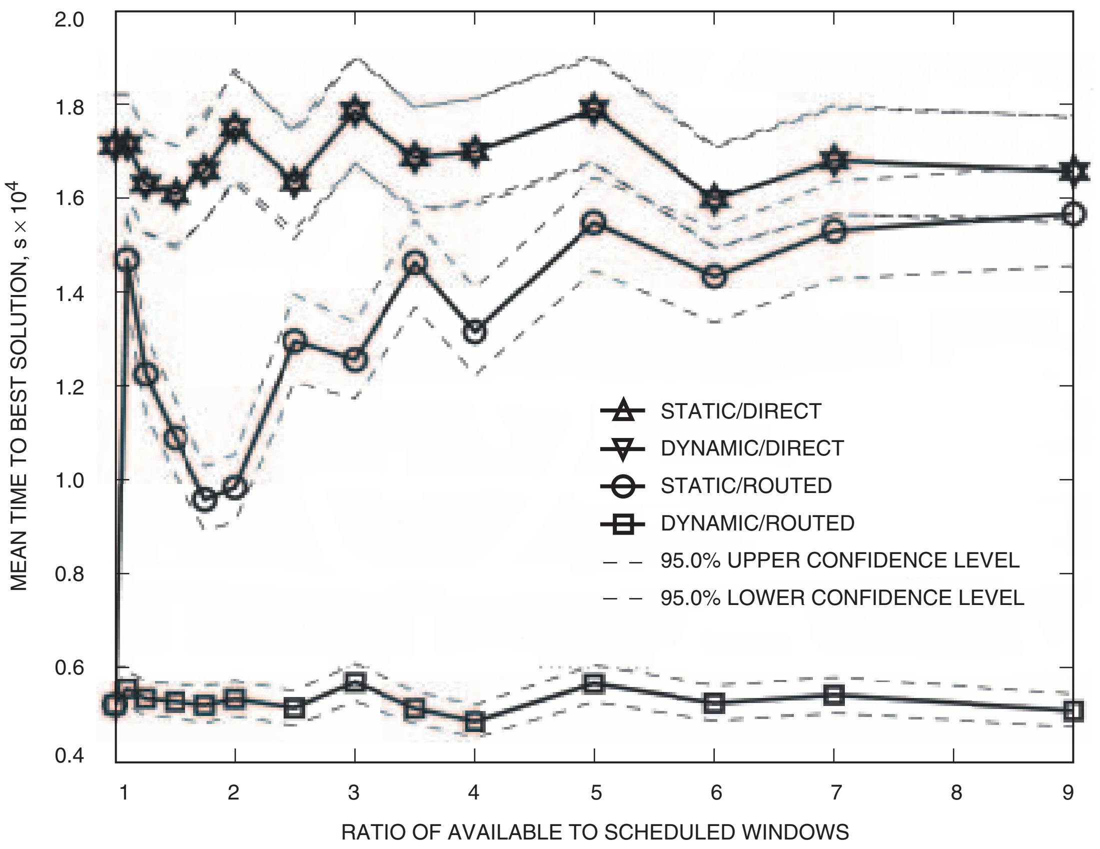

Fig. 5. Improvement of adaptive rescheduling in delivery time from a rover to Earth.

rover's data in the above experiments. However, once they have agreed to the new communications goal, the network of spacecraft may need to reschedule other communications. But, how long does it take for the network to reach consensus on a new schedule based on the addition of a goal?

We performed experiments for the Mars network given the same communication scheme in Fig. 4 to see how long it would take the distributed network scheduler to reach consensus upon the addition of a new communications goal. The passing of time was simulated, and coordination messages were routed through SHAC according to the availability communication windows. We used the SHAC model in Section V.B to coordinate scheduling. To simulate the slowdown of flight processors, we need time to pass around 500 to 1000 times faster than real time. Since all five spacecraft were run on the same workstation in these experiments, we simulated time passing 100 times faster than system time for a slowdown factor of 500, roughly. Measurements were made in both central processing unit (CPU) seconds and actual seconds while simulating time. Replanning cycles (1 for each SHAC cycle) were restricted to 20 ASPEN iterations, and when there were no remaining conflicts, SHAC slept for 0.1 second (10 simulated seconds) between cycles. Results for 5 problems are reported for each number of goals ranging between 3 and 20. These are numbers of communication goals for each spacecraft over a 3-day horizon. Goals were randomly generated with data sizes ranging between 2 and 100 kbits, with delivery deadlines ranging between 40,000 and 100,000 seconds (approximately 11 to 28 hours), and with random destinations to either Earth or a rover (but rovers always send to Earth). Once a schedule was generated, another goal was added for a random spacecraft at a random time in order to measure the effort in reaching consensus for an unexpected event.

Figures 6 through 9 show the results of rescheduling for one goal added to the system. Each point in the plots is a mean, minimum, or maximum value for the five spacecraft. Figure 6 shows that CPU time is limited to less than 200 seconds for a spacecraft. On a flight processor, this translates to 100,000 to 200,000 seconds, or 28 to 56 hours. The mean maximum time is from 4 to 30 seconds, or 1 to 14 hours. The system time in Fig. 7 shows that, on average, the system reaches consensus in less than an hour (according to the maximum time trend line), but it can take a little more than 200 actual seconds in some cases. We must multiply this by the simulation speedup factor of 100; thus, the approximate maximum time onboard for this problem set would be roughly 56 hours. This means that, without careful coordination, activities scheduled within 56 hours of the unexpected event could be executed inconsistently with respect to other spacecraft. To resolve this problem, the spacecraft should switch to a simpler protocol with real-time consensus guarantees for activities soon to be executed. Although not employed in this application, an algorithm for determining when this switch should occur (based on the time of the activity, communication windows, and the routing required by the simpler protocol) is described in [2]. Figures 8 and 9 show the communication overhead for reaching consensus. Many of the goals could be resolved locally, requiring no communication.

The generation of initial schedules demonstrates collaborative ground planning capability. During this generation, messages sent through SHAC were not restricted to communication window constraints as they were for the addition of a goal in the results above. Results for distributed scheduling performance are given in Figs. 10 through 14. These timing results are for reaching consensus on the scheduled goals. The plots on message numbers and data size are of scheduling communications through SHAC (not the image data). Trend lines are exponential, although it has not been determined whether the algorithms explore a search space that grows exponentially with problem size. Figure 11 shows that the processing overhead for SHAC is significant (in some cases more than 50 percent). This is because SHAC is often looping while waiting for other spacecraft to replan and send out modifications. This looping is necessary when scheduling during execution in order to respond to unexpected events. However, the bookkeeping done in this loop could be better streamlined. Otherwise, overhead could be reduced by lengthening the replanning cycle and increasing the sleep time between SHAC cycles. Figure 12 shows that the time to reach consensus on generating a schedule ranges from seconds to 50 minutes.

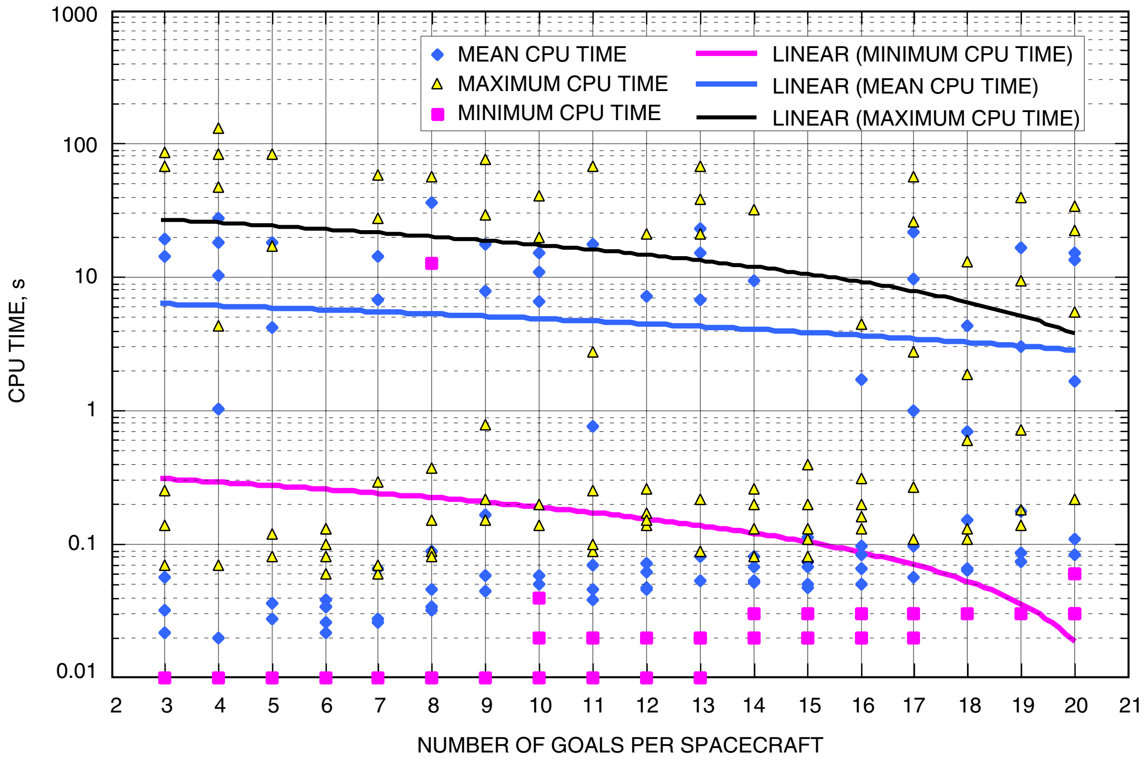

Fig. 6. CPU time for scheduling a single goal.

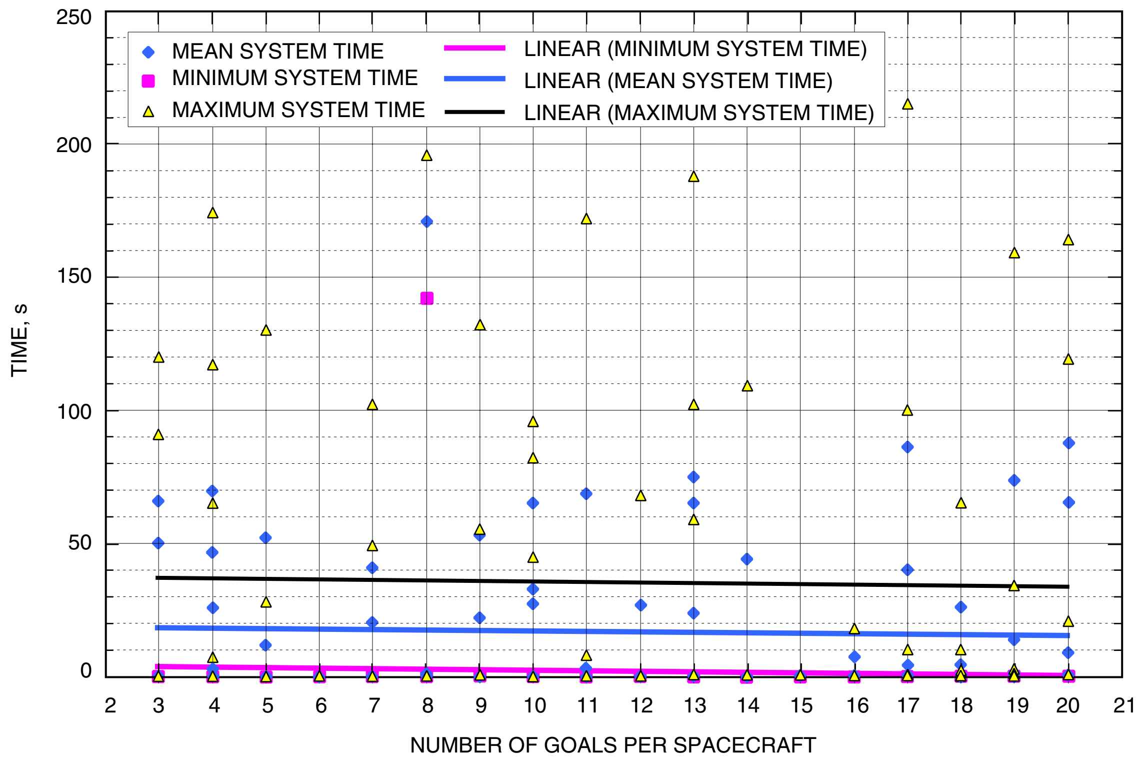

Fig. 7. Actual time to schedule a single goal.

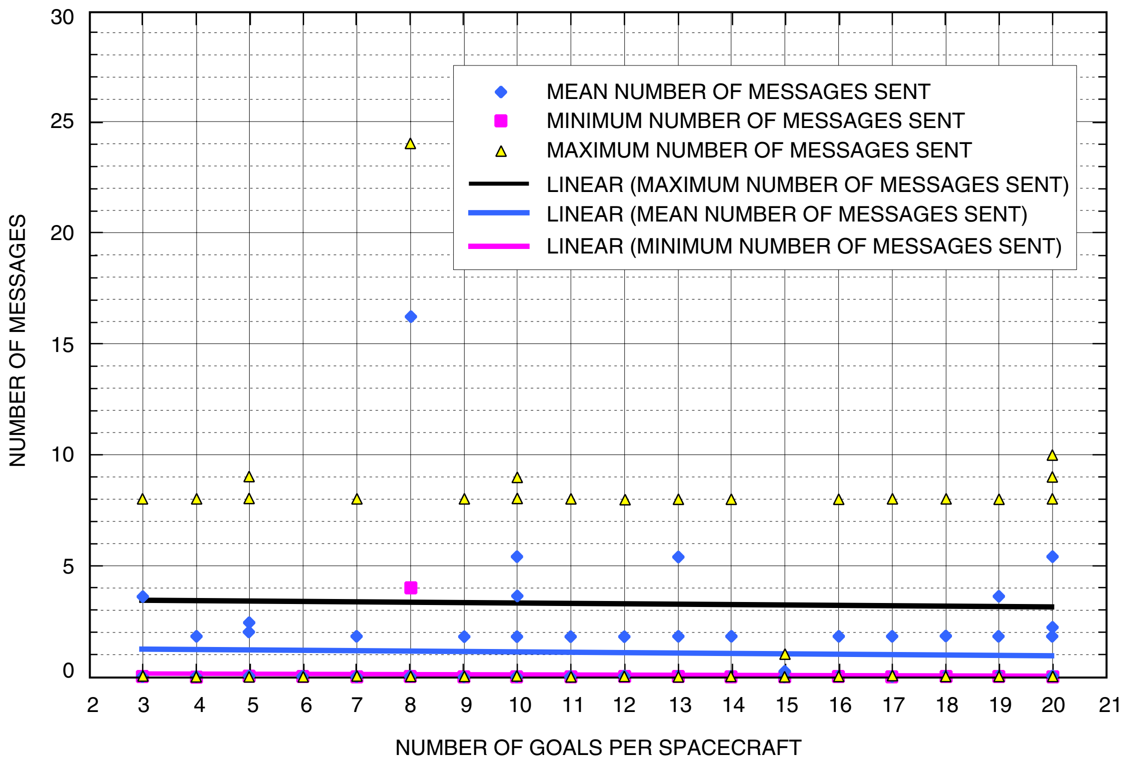

Fig. 8. Number of messages sent for scheduling a single goal.

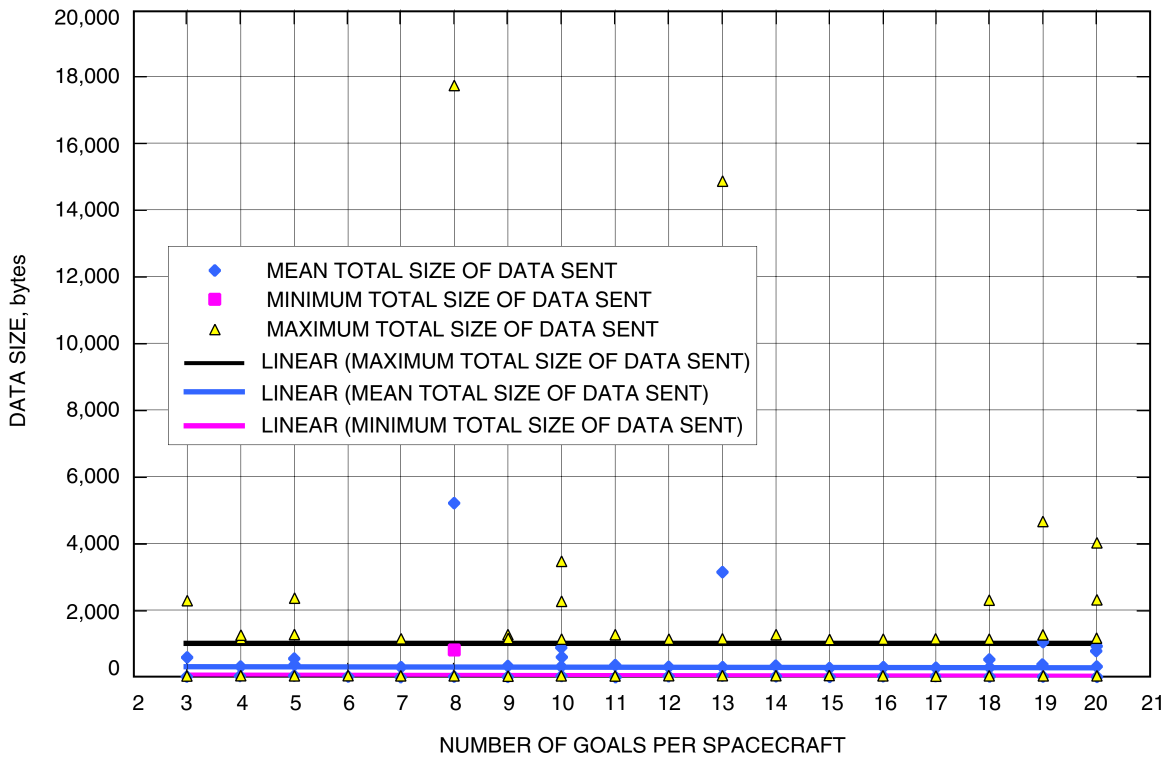

Fig. 9. Total size of data sent for scheduling a single goal.

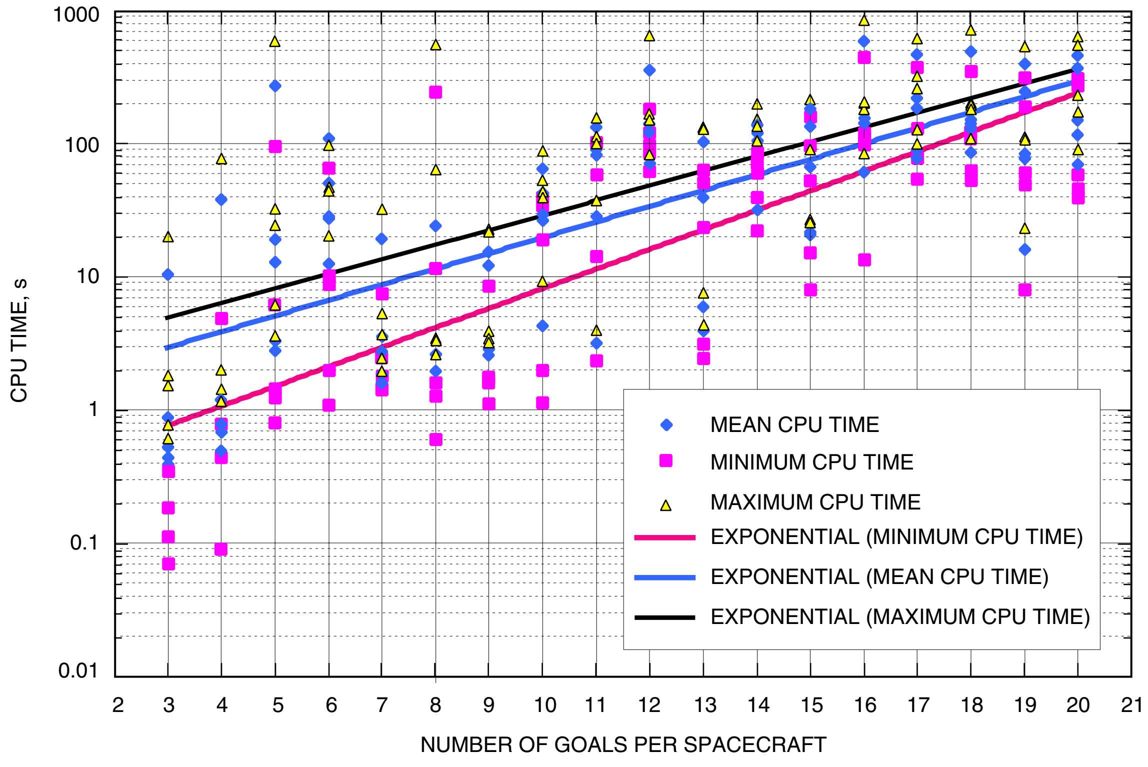

Fig. 10. CPU time for schedule generation.

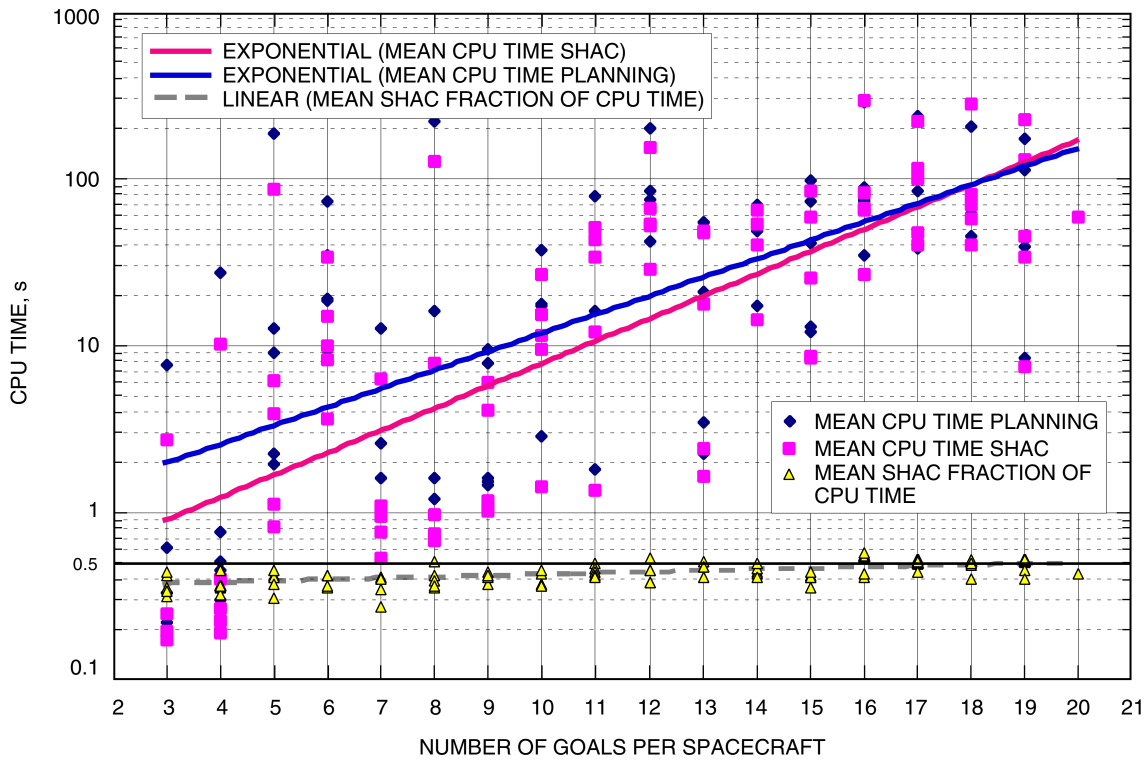

Fig. 11. CPU time of SHAC versus replanning for schedule generation.

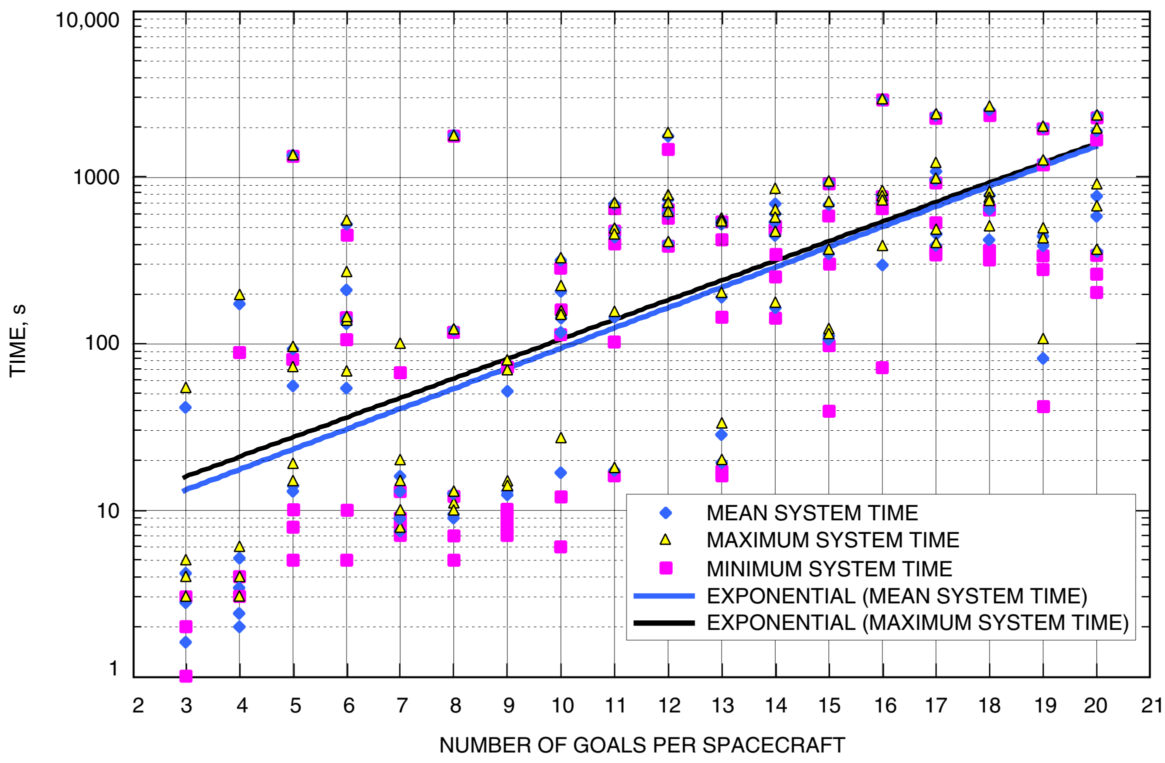

Fig. 12. Actual time for generating schedules.

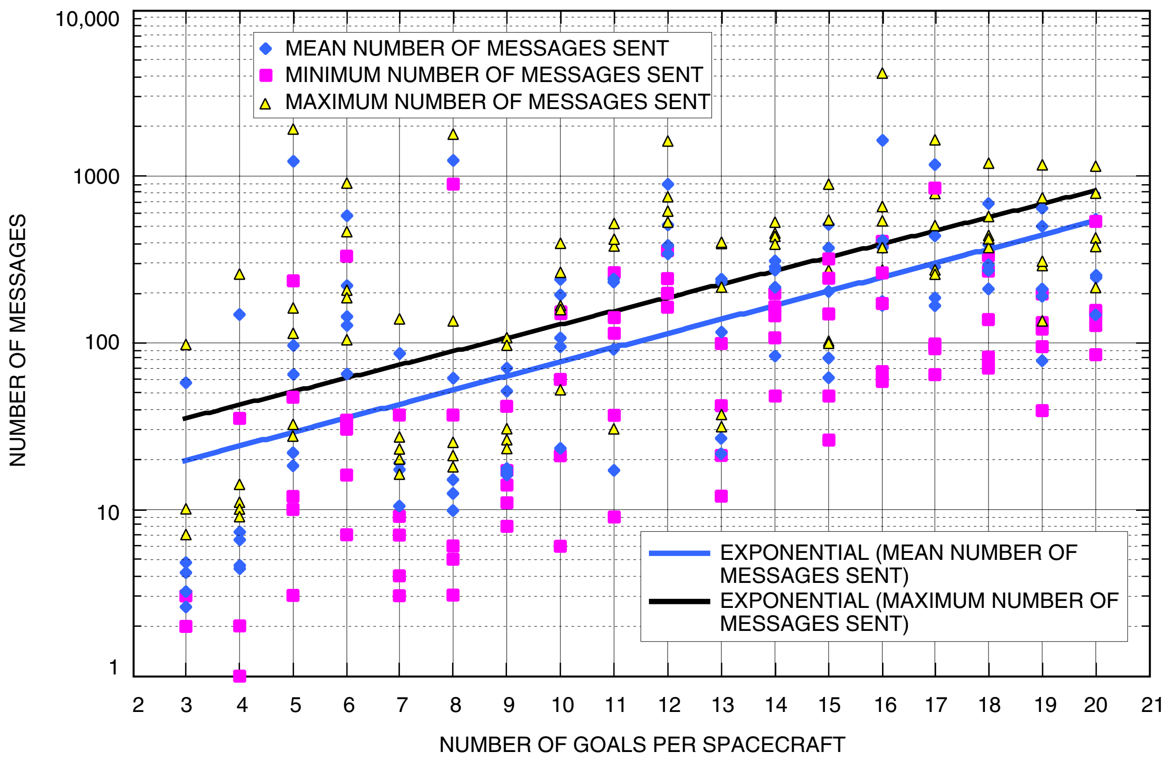

Fig. 13. Number of messages sent in generating schedules.

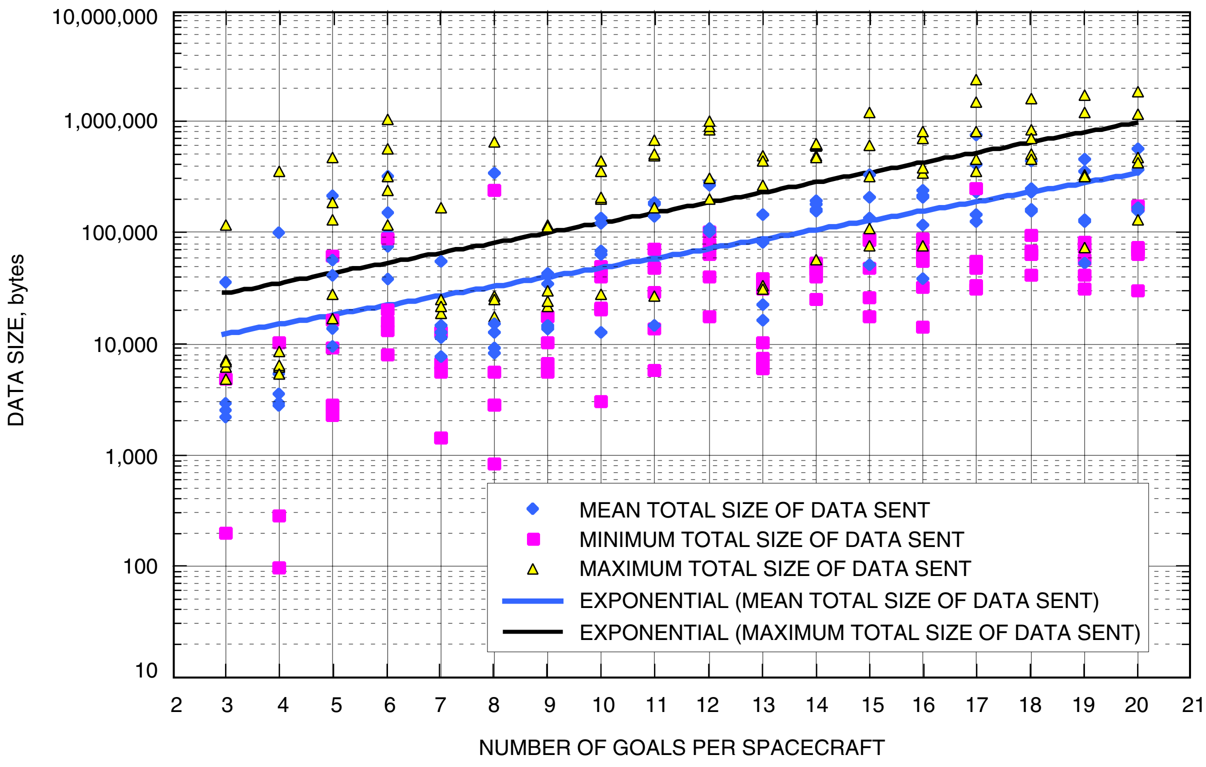

Fig. 14. Total size of data sent while generating schedules.

VII. Conclusion

This article described software for onboard distributed scheduling of communications among a network of spacecraft. The distributed network scheduler enables reactive communications within the context of scheduled operations by autonomously negotiating communication changes. In addition, the scheduling system can generate schedules using the same negotiation mechanisms to enable missions (such as the many for Mars) to collaboratively schedule communications on the ground. Evaluations were based on models of MER-A, MER-B, MGS, Odyssey, and Mars Express. Our main results include

(1) A simulation showing a maximum opportunity of improving the traverse exploration rate by a factor of three

(2) A simulation showing reduction in one-way delivery times from a rover to Earth from as much as 5 hours to 1.5 hours

(3) Simulated onboard response to unexpected events averages under an hour

(4) Ground schedule generation ranging from seconds to 50 minutes for 15 to 100 goals

Future work includes extending the software to schedule during execution using various protocols to give real-time consensus guarantees and integrating the software with a realistic communications simulator.

Acknowledgment

This work was performed for the Intelligent Network Management Thrust of JPL's Interplanetary Network Directorate. As manager of the thrust, Loren Clare provided input throughout development on the modeling of communications activities.

References

[1] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau, "Using Iterative Repair to Improve Responsiveness of Planning and Scheduling," 5th International Conference on Artificial Intelligence Planning Systems (AIPS 2000), Breckenridge, Colorado, April 2000.

[2] B. Clement and A. Barrett, "Continual Coordination through Shared Activities," 2nd International Conference on Autonomous and Multi-Agent Systems (AAMAS 2003), Melbourne, Australia, July 2003.

[3] R. Sherwood, B. Engelhardt, G. Rabideau, S. Chien, and R. Knight, *ASPEN User's Guide*, JPL D-15482, Jet Propulsion Laboratory, Pasadena, California, February 28, 2000.
http://www-aig.jpl.nasa.gov/public/planning/aspen/